注: 1.阅读前你需要知道java的反射原理 2.spring的BeanPostProcessor后置处理器

本章介绍一个spring中的特别重要的接口BeanPostProcessor后置处理器 后置处理器简单来说就是一个拦截器,spring就是使用这个来对bean进行功能的增强，比如说:aop是实现就是使用后置处理器 tx 同样也是使用这种策略 但是每一种后置处理器的调用时机都略有不同，但总体是来做bean功能增强的 本篇请先关心这几种 BeanPostProcessor、InstantiationAwareBeanPostProcessor、 MergedBeanDefinitionPostProcessor XmlBeanFactory总体来说分两步: 1.bean信息的注册 1.定位 定位bean信息 2.载入并解析 解析bean的配置信息 3.注册 注册到工厂中 2.bean的实例生成 1.拿到bean的信息BeanDefinition 2.=工厂创建bean的实例 第一步的bean信息的注册过程已经在上篇中讲完了 本篇主要讲第二步:bean的实例生成

解析入口测试代码:

```
ClassPathResource pathResource = new ClassPathResource("/bean.xml");
XmlBeanFactory beanFactory = new XmlBeanFactory(pathResource);
Object person = beanFactory.getBean("person");
```

主要是解析Object person = beanFactory.getBean("person")的过程

解析开始: 1.doGetBean(name, null, null, false);

```
//这一段的信息量爆炸,需要仔细分析
protected <T> T doGetBean(
            final String name, final Class<T> requiredType, final Object[] args, boolean
typeCheckOnly)
            throws BeansException {

    final String beanName = transformedBeanName(name);
    Object bean;

    // Eagerly check singleton cache for manually registered singletons.
    //先从缓存中尝试去获取这个bean的实例:缓存时个Map(private final Map<String, Object> singletonObjects =
new ConcurrentHashMap<String, Object>(256))
    Object sharedInstance = getSingleton(beanName);
    if (sharedInstance != null && args == null) {
        if (logger.isDebugEnabled()) {
            if (isSingletonCurrentlyInCreation(beanName)) {
                logger.debug("Returning eagerly cached instance of singleton bean '" + beanName +
                        "' that is not fully initialized yet - a consequence of a circular
reference");
            }
            else {
                logger.debug("Returning cached instance of singleton bean '" + beanName + "'");
            }
        }
        bean = getObjectForBeanInstance(sharedInstance, name, beanName, null);
    }

    else {
        // Fail if we're already creating this bean instance:
        // We're assumably within a circular reference.
        //判断当前bean实例是否正在被创建中,防多线程重复创建
        if (isPrototypeCurrentlyInCreation(beanName)) {
            throw new BeanCurrentlyInCreationException(beanName);
```

```
        }

        // Check if bean definition exists in this factory.
        BeanFactory parentBeanFactory = getParentBeanFactory();
        if (parentBeanFactory != null && !containsBeanDefinition(beanName)) {
            // Not found -> check parent.
            String nameToLookup = originalBeanName(name);
            if (args != null) {
                // Delegation to parent with explicit args.
                return (T) parentBeanFactory.getBean(nameToLookup, args);
            }
            else {
                // No args -> delegate to standard getBean method.
                return parentBeanFactory.getBean(nameToLookup, requiredType);
            }
        }
        //标记当前bean已经被创建
        if (!typeCheckOnly) {
            markBeanAsCreated(beanName);
        }

        try {
            //获取这个bean的定义信息BeanDefinition，并包装成RootBeanDefinition
            final RootBeanDefinition mbd = getMergedLocalBeanDefinition(beanName);
            checkMergedBeanDefinition(mbd, beanName, args);

            // Guarantee initialization of beans that the current bean depends on.
            //获取当前Bean依赖的其他Bean;如果有按照getBean()把依赖的Bean先创建出来
            String[] dependsOn = mbd.getDependsOn();
            if (dependsOn != null) {
                for (String dep : dependsOn) {
                    if (isDependent(beanName, dep)) {
                        throw new BeanCreationException(mbd.getResourceDescription(), beanName,
                                "Circular depends-on relationship between '" + beanName + "' and
'" + dep + "'");
                    }
                    registerDependentBean(dep, beanName);
                    getBean(dep);
                }
            }

            // Create bean instance.
            if (mbd.isSingleton()) {
                //启动单实例Bean的创建流程
                sharedInstance = getSingleton(beanName, new ObjectFactory<Object>() {
                    @Override
                    public Object getObject() throws BeansException {
                        try {
                            return createBean(beanName, mbd, args);
                        }
                        catch (BeansException ex) {
                            // Explicitly remove instance from singleton cache: It might have been
put there
                            // eagerly by the creation process, to allow for circular reference
resolution.
                            // Also remove any beans that received a temporary reference to the
bean.
```

```
                            destroySingleton(beanName);
                            throw ex;
                        }
                    }
                });
                bean = getObjectForBeanInstance(sharedInstance, name, beanName, mbd);
            }
            //创建多实例bean
            else if (mbd.isPrototype()) {
                // It's a prototype -> create a new instance.
                Object prototypeInstance = null;
                try {
                    beforePrototypeCreation(beanName);
                    prototypeInstance = createBean(beanName, mbd, args);
                }
                finally {
                    afterPrototypeCreation(beanName);
                }
                bean = getObjectForBeanInstance(prototypeInstance, name, beanName, mbd);
            }

            else {

                String scopeName = mbd.getScope();
                final Scope scope = this.scopes.get(scopeName);
                if (scope == null) {
                    throw new IllegalStateException("No Scope registered for scope name '" +
scopeName + "'");
                }
                try {
                    Object scopedInstance = scope.get(beanName, new ObjectFactory<Object>() {
                        @Override
                        public Object getObject() throws BeansException {
                            beforePrototypeCreation(beanName);
                            try {
                                return createBean(beanName, mbd, args);
                            }
                            finally {
                                afterPrototypeCreation(beanName);
                            }
                        }
                    });
                    bean = getObjectForBeanInstance(scopedInstance, name, beanName, mbd);
                }
                catch (IllegalStateException ex) {
                    throw new BeanCreationException(beanName,
                            "Scope '" + scopeName + "' is not active for the current thread;
consider " +
                            "defining a scoped proxy for this bean if you intend to refer to it
from a singleton",
                            ex);
                }
            }
        }
        catch (BeansException ex) {
            cleanupAfterBeanCreationFailure(beanName);
            throw ex;
```

```
        }
    }

    // Check if required type matches the type of the actual bean instance.
    if (requiredType != null && bean != null && !requiredType.isInstance(bean)) {
        try {
            return getTypeConverter().convertIfNecessary(bean, requiredType);
        }
        catch (TypeMismatchException ex) {
            if (logger.isDebugEnabled()) {
                logger.debug("Failed to convert bean '" + name + "' to required type '" +
                        ClassUtils.getQualifiedName(requiredType) + "'", ex);
            }
            throw new BeanNotOfRequiredTypeException(name, requiredType, bean.getClass());
        }
    }
    return (T) bean;
}
```

以下步骤是对上面这段代码的分析结果简要摘取: 2、先获取缓存中保存的单实例Bean。如果能获取到说明这个Bean之前被创建过（所有创建过的单实例Bean都会被缓存起来） 从private final Map<String, Object> singletonObjects = new ConcurrentHashMap<String, Object>(256);获取的 3、缓存中获取不到，开始Bean的创建对象流程； 4、标记当前bean已经被创建 -> markBeanAsCreated(beanName) 防止对线程重复创建 5.获取需要创建bean的定义信息BeanDefinition，并包装成RootBeanDefinition

```
protected RootBeanDefinition getMergedLocalBeanDefinition(String beanName) throws BeansException {
    // Quick check on the concurrent map first, with minimal locking.
    //如果可以拿到就直接返回
    RootBeanDefinition mbd = this.mergedBeanDefinitions.get(beanName);
    if (mbd != null) {
        return mbd;
    }
    return getMergedBeanDefinition(beanName, getBeanDefinition(beanName));
}
public BeanDefinition getBeanDefinition(String beanName) throws NoSuchBeanDefinitionException {
    //实际上也是从我们上篇分析中的那个beanDefinitionMap中获取
    BeanDefinition bd = this.beanDefinitionMap.get(beanName);
    if (bd == null) {
        if (this.logger.isTraceEnabled()) {
            this.logger.trace("No bean named '" + beanName + "' found in " + this);
        }
        throw new NoSuchBeanDefinitionException(beanName);
    }
    return bd;
}
```

6. 【获取当前Bean依赖的其他Bean;如果有按照getBean()把依赖的Bean先创建出来】
7. 启动单实例Bean的创建流程；

    1）、createBean(beanName, mbd, args);

```
sharedInstance = getSingleton(beanName, new ObjectFactory<Object>() {
    @Override
    public Object getObject() throws BeansException {
        try {
```

```
            return createBean(beanName, mbd, args);
        }
        catch (BeansException ex) {
            // Explicitly remove instance from singleton cache: It might have been put there
            // eagerly by the creation process, to allow for circular reference resolution.
            // Also remove any beans that received a temporary reference to the bean.
            destroySingleton(beanName);
            throw ex;
        }
    }
});
singletonObject = singletonFactory.getObject() ->  createBean(beanName, mbd, args)(实际上调用的是子类的方法)
//createBean ()
protected Object createBean(String beanName, RootBeanDefinition mbd, Object[] args) throws BeanCreationException {
    if (logger.isDebugEnabled()) {
        logger.debug("Creating instance of bean '" + beanName + "'");
    }
    RootBeanDefinition mbdToUse = mbd;

    // Make sure bean class is actually resolved at this point, and
    // clone the bean definition in case of a dynamically resolved Class
    // which cannot be stored in the shared merged bean definition.
    Class<?> resolvedClass = resolveBeanClass(mbd, beanName);
    if (resolvedClass != null && !mbd.hasBeanClass() && mbd.getBeanClassName() != null) {
        mbdToUse = new RootBeanDefinition(mbd);
        mbdToUse.setBeanClass(resolvedClass);
    }

    // Prepare method overrides.
    try {
        mbdToUse.prepareMethodOverrides();
    }
    catch (BeanDefinitionValidationException ex) {
        throw new BeanDefinitionStoreException(mbdToUse.getResourceDescription(),
                beanName, "Validation of method overrides failed", ex);
    }

    try {
        // Give BeanPostProcessors a chance to return a proxy instead of the target bean instance.
        //创建对象前先尝试返回一个代理对象
        Object bean = resolveBeforeInstantiation(beanName, mbdToUse);
        if (bean != null) {
            return bean;
        }
    }
    catch (Throwable ex) {
        throw new BeanCreationException(mbdToUse.getResourceDescription(), beanName,
                "BeanPostProcessor before instantiation of bean failed", ex);
    }
    //如果没有返回代理对象就创建
    Object beanInstance = doCreateBean(beanName, mbdToUse, args);
    if (logger.isDebugEnabled()) {
        logger.debug("Finished creating instance of bean '" + beanName + "'");
    }
```

```
    return beanInstance;
}
```

2）、Object bean = resolveBeforeInstantiation(beanName, mbdToUse);让BeanPostProcessor先拦截返回代理对象； 源码注释:// Give BeanPostProcessors a chance to return a proxy instead of the target bean instance 这句最能表达愿意特列在此处 【InstantiationAwareBeanPostProcessor】：提前执行； 先触发：postProcessBeforeInstantiation()； 如果有返回值：触发postProcessAfterInitialization()；

3）、如果前面的InstantiationAwareBeanPostProcessor没有返回代理对象；调用4） 4）、Object beanInstance = doCreateBean(beanName, mbdToUse, args);创建Bean

```
protected Object doCreateBean(final String beanName, final RootBeanDefinition mbd, final
Object[] args)
            throws BeanCreationException {

    // Instantiate the bean.
    BeanWrapper instanceWrapper = null;
    if (mbd.isSingleton()) {
        instanceWrapper = this.factoryBeanInstanceCache.remove(beanName);
    }
    if (instanceWrapper == null) {
        /**
        1）、【创建Bean实例】; createBeanInstance(beanName, mbd, args);
            注意:此时只是实例刚被创建出来而已但是实例的字段什么的还未被初始化
            利用工厂方法或者对象的构造器创建出Bean实例 使用反射进行实例的创建
        */
        instanceWrapper = createBeanInstance(beanName, mbd, args);
    }
    final Object bean = (instanceWrapper != null ? instanceWrapper.getWrappedInstance() :
null);
    Class<?> beanType = (instanceWrapper != null ? instanceWrapper.getWrappedClass() : null);
    mbd.resolvedTargetType = beanType;

    // Allow post-processors to modify the merged bean definition.
    synchronized (mbd.postProcessingLock) {
        if (!mbd.postProcessed) {
            try {
                /**
                2）、applyMergedBeanDefinitionPostProcessors(mbd, beanType, beanName);
            调用MergedBeanDefinitionPostProcessor的postProcessMergedBeanDefinition(mbd,
beanType, beanName);
                */
                applyMergedBeanDefinitionPostProcessors(mbd, beanType, beanName);
            }
            catch (Throwable ex) {
                throw new BeanCreationException(mbd.getResourceDescription(), beanName,
                        "Post-processing of merged bean definition failed", ex);
            }
            mbd.postProcessed = true;
        }
    }

    // Eagerly cache singletons to be able to resolve circular references
    // even when triggered by lifecycle interfaces like BeanFactoryAware.
    boolean earlySingletonExposure = (mbd.isSingleton() && this.allowCircularReferences &&
            isSingletonCurrentlyInCreation(beanName));
```

```
    if (earlySingletonExposure) {
        if (logger.isDebugEnabled()) {
            logger.debug("Eagerly caching bean '" + beanName +
                    "' to allow for resolving potential circular references");
        }
        addSingletonFactory(beanName, new ObjectFactory<Object>() {
            @Override
            public Object getObject() throws BeansException {
                return getEarlyBeanReference(beanName, mbd, bean);
            }
        });
    }

    // Initialize the bean instance.
    Object exposedObject = bean;
    try {
        /**
        3) 、【Bean属性赋值】populateBean(beanName, mbd, instanceWrapper);
        赋值之前:
            1) 、拿到InstantiationAwareBeanPostProcessor后置处理器;
                postProcessAfterInstantiation();
            2) 、拿到InstantiationAwareBeanPostProcessor后置处理器;
                postProcessPropertyValues();
            =====赋值之前: ===
            3) 、应用Bean属性的值;为属性利用setter方法等进行赋值;
                applyPropertyValues(beanName, mbd, bw, pvs);
        */
        populateBean(beanName, mbd, instanceWrapper);
        if (exposedObject != null) {
            /**
            4) 、【Bean初始化】initializeBean(beanName, exposedObject, mbd);
            */
            exposedObject = initializeBean(beanName, exposedObject, mbd);
        }
    }
    catch (Throwable ex) {
        if (ex instanceof BeanCreationException && beanName.equals(((BeanCreationException)
ex).getBeanName())) {
            throw (BeanCreationException) ex;
        }
        else {
            throw new BeanCreationException(
                    mbd.getResourceDescription(), beanName, "Initialization of bean failed",
ex);
        }
    }

    if (earlySingletonExposure) {
        Object earlySingletonReference = getSingleton(beanName, false);
        if (earlySingletonReference != null) {
            if (exposedObject == bean) {
                exposedObject = earlySingletonReference;
            }
            else if (!this.allowRawInjectionDespiteWrapping && hasDependentBean(beanName)) {
                String[] dependentBeans = getDependentBeans(beanName);
                Set<String> actualDependentBeans = new LinkedHashSet<String>
(dependentBeans.length);
```

```
                for (String dependentBean : dependentBeans) {
                    if (!removeSingletonIfCreatedForTypeCheckOnly(dependentBean)) {
                        actualDependentBeans.add(dependentBean);
                    }
                }
                if (!actualDependentBeans.isEmpty()) {
                    throw new BeanCurrentlyInCreationException(beanName,
                            "Bean with name '" + beanName + "' has been injected into other
beans [" +
                            StringUtils.collectionToCommaDelimitedString(actualDependentBeans)
+
                            "] in its raw version as part of a circular reference, but has
eventually been " +
                            "wrapped. This means that said other beans do not use the final
version of the " +
                            "bean. This is often the result of over-eager type matching -
consider using " +
                            "'getBeanNamesOfType' with the 'allowEagerInit' flag turned off,
for example.");
                }
            }
        }
    }

    // Register bean as disposable.
    try {
        /**
        5) 、注册Bean的销毁方法;
        */
        registerDisposableBeanIfNecessary(beanName, bean, mbd);
    }
    catch (BeanDefinitionValidationException ex) {
        throw new BeanCreationException(
                mbd.getResourceDescription(), beanName, "Invalid destruction signature", ex);
    }

    return exposedObject;
}
//1) 、【创建Bean实例】; createBeanInstance(beanName, mbd, args);
protected BeanWrapper createBeanInstance(String beanName, RootBeanDefinition mbd, Object[]
args) {
    // Make sure bean class is actually resolved at this point.
    Class<?> beanClass = resolveBeanClass(mbd, beanName);

    if (beanClass != null && !Modifier.isPublic(beanClass.getModifiers()) &&
!mbd.isNonPublicAccessAllowed()) {
        throw new BeanCreationException(mbd.getResourceDescription(), beanName,
                "Bean class isn't public, and non-public access not allowed: " +
beanClass.getName());
    }

    if (mbd.getFactoryMethodName() != null)  {
        //使用工厂方法创建实例
        return instantiateUsingFactoryMethod(beanName, mbd, args);
    }

    // Shortcut when re-creating the same bean...
```

```
    boolean resolved = false;
    boolean autowireNecessary = false;
    if (args == null) {
        synchronized (mbd.constructorArgumentLock) {
            if (mbd.resolvedConstructorOrFactoryMethod != null) {
                resolved = true;
                autowireNecessary = mbd.constructorArgumentsResolved;
            }
        }
    }
    if (resolved) {
        if (autowireNecessary) {
            //使用构造器进行创建实例
            return autowireConstructor(beanName, mbd, null, null);
        }
        else {
            return instantiateBean(beanName, mbd);
        }
    }

    // Need to determine the constructor...
    Constructor<?>[] ctors = determineConstructorsFromBeanPostProcessors(beanClass, beanName);
    if (ctors != null ||
            mbd.getResolvedAutowireMode() == RootBeanDefinition.AUTOWIRE_CONSTRUCTOR ||
            mbd.hasConstructorArgumentValues() || !ObjectUtils.isEmpty(args))  {
        //使用构造器进行创建实例
        return autowireConstructor(beanName, mbd, ctors, args);
    }

    // No special handling: simply use no-arg constructor.
    //无参构造器创建实例
    return instantiateBean(beanName, mbd);
}
//2) 、applyMergedBeanDefinitionPostProcessors(mbd, beanType, beanName);
protected void applyMergedBeanDefinitionPostProcessors(RootBeanDefinition mbd, Class<?>
beanType, String beanName) {
    //循环调用注册在容器中的后置处理器:BeanPostProcessor.postProcessBeforeInitialization () ;
    for (BeanPostProcessor bp : getBeanPostProcessors()) {
        if (bp instanceof MergedBeanDefinitionPostProcessor) {
            MergedBeanDefinitionPostProcessor bdp = (MergedBeanDefinitionPostProcessor) bp;
            bdp.postProcessMergedBeanDefinition(mbd, beanType, beanName);
        }
    }
}
//3) 、【Bean属性赋值】populateBean(beanName, mbd, instanceWrapper);
protected void populateBean(String beanName, RootBeanDefinition mbd, BeanWrapper bw) {
    PropertyValues pvs = mbd.getPropertyValues();

    if (bw == null) {
        if (!pvs.isEmpty()) {
            throw new BeanCreationException(
                    mbd.getResourceDescription(), beanName, "Cannot apply property values to
null instance");
        }
        else {
            // Skip property population phase for null instance.
            return;
```

```
        }
    }

    // Give any InstantiationAwareBeanPostProcessors the opportunity to modify the
    // state of the bean before properties are set. This can be used, for example,
    // to support styles of field injection.
    boolean continueWithPropertyPopulation = true;

    /**
    拿到InstantiationAwareBeanPostProcessor后置处理器;
                postProcessAfterInstantiation();
    */
    if (!mbd.isSynthetic() && hasInstantiationAwareBeanPostProcessors()) {
        for (BeanPostProcessor bp : getBeanPostProcessors()) {
            if (bp instanceof InstantiationAwareBeanPostProcessor) {
                InstantiationAwareBeanPostProcessor ibp =
(InstantiationAwareBeanPostProcessor) bp;
                if (!ibp.postProcessAfterInstantiation(bw.getWrappedInstance(), beanName)) {
                    continueWithPropertyPopulation = false;
                    break;
                }
            }
        }
    }

    if (!continueWithPropertyPopulation) {
        return;
    }

    if (mbd.getResolvedAutowireMode() == RootBeanDefinition.AUTOWIRE_BY_NAME ||
            mbd.getResolvedAutowireMode() == RootBeanDefinition.AUTOWIRE_BY_TYPE) {
        MutablePropertyValues newPvs = new MutablePropertyValues(pvs);

        // Add property values based on autowire by name if applicable.
        if (mbd.getResolvedAutowireMode() == RootBeanDefinition.AUTOWIRE_BY_NAME) {
            autowireByName(beanName, mbd, bw, newPvs);
        }

        // Add property values based on autowire by type if applicable.
        if (mbd.getResolvedAutowireMode() == RootBeanDefinition.AUTOWIRE_BY_TYPE) {
            autowireByType(beanName, mbd, bw, newPvs);
        }

        pvs = newPvs;
    }

    boolean hasInstAwareBpps = hasInstantiationAwareBeanPostProcessors();
    boolean needsDepCheck = (mbd.getDependencyCheck() !=
RootBeanDefinition.DEPENDENCY_CHECK_NONE);

    if (hasInstAwareBpps || needsDepCheck) {
        PropertyDescriptor[] filteredPds = filterPropertyDescriptorsForDependencyCheck(bw,
mbd.allowCaching);
        if (hasInstAwareBpps) {
            /**
            拿到InstantiationAwareBeanPostProcessor后置处理器;
                postProcessPropertyValues();
```

```
            */
            for (BeanPostProcessor bp : getBeanPostProcessors()) {
                if (bp instanceof InstantiationAwareBeanPostProcessor) {
                    InstantiationAwareBeanPostProcessor ibp =
(InstantiationAwareBeanPostProcessor) bp;
                    pvs = ibp.postProcessPropertyValues(pvs, filteredPds,
bw.getWrappedInstance(), beanName);
                    if (pvs == null) {
                        return;
                    }
                }
            }
        }
        if (needsDepCheck) {
            checkDependencies(beanName, mbd, filteredPds, pvs);
        }
    }
    /**
    应用Bean属性的值；为属性利用setter方法等进行赋值；
    applyPropertyValues(beanName, mbd, bw, pvs);
    */
    applyPropertyValues(beanName, mbd, bw, pvs);
}
//4)、【Bean初始化】initializeBean(beanName, exposedObject, mbd);
protected Object initializeBean(final String beanName, final Object bean, RootBeanDefinition
mbd) {
    if (System.getSecurityManager() != null) {
        AccessController.doPrivileged(new PrivilegedAction<Object>() {
            @Override
            public Object run() {
                invokeAwareMethods(beanName, bean);
                return null;
            }
        }, getAccessControlContext());
    }
    else {
        /**
        1)、【执行Aware接口方法】invokeAwareMethods(beanName, bean);执行xxxAware接口的方法
            BeanNameAware\BeanClassLoaderAware\BeanFactoryAware
        */
        invokeAwareMethods(beanName, bean);
    }

    Object wrappedBean = bean;
    if (mbd == null || !mbd.isSynthetic()) {
        /**
        2)、【执行后置处理器初始化之前】applyBeanPostProcessorsBeforeInitialization(wrappedBean,
beanName);
            BeanPostProcessor.postProcessBeforeInitialization () ;
        */
        wrappedBean = applyBeanPostProcessorsBeforeInitialization(wrappedBean, beanName);
    }

    try {
        /**
        3)、【执行初始化方法】invokeInitMethods(beanName, wrappedBean, mbd);
            1)、是否是InitializingBean接口的实现；执行接口规定的初始化；
```

```
                示例如下：
                @Component
                public class Cat implements InitializingBean {

                    public Cat(){
                        System.out.println("cat constructor...");
                    }

                    @Override
                    public void afterPropertiesSet() throws Exception {
                        // TODO Auto-generated method stub
                        System.out.println("cat...afterPropertiesSet...");
                    }
                }
                如果bean中定义了则回调afterPropertiesSet方法来进行初始化
                2）、是否自定义初始化方法；可通过@Bean注解中的initMethod 配置文件的xml定义的初始化方法等
            */
            invokeInitMethods(beanName, wrappedBean, mbd);
        }
        catch (Throwable ex) {
            throw new BeanCreationException(
                    (mbd != null ? mbd.getResourceDescription() : null),
                    beanName, "Invocation of init method failed", ex);
        }

        if (mbd == null || !mbd.isSynthetic()) {
            /**
            4）、【执行后置处理器初始化之后】applyBeanPostProcessorsAfterInitialization
                    BeanPostProcessor.postProcessAfterInitialization();
            */
            wrappedBean = applyBeanPostProcessorsAfterInitialization(wrappedBean, beanName);
        }
        return wrappedBean;
    }
```

5）、将创建的Bean添加到缓存中singletonObjects； addSingleton(beanName, singletonObject);

```
public Object getSingleton(String beanName, ObjectFactory<?> singletonFactory) {
    Assert.notNull(beanName, "'beanName' must not be null");
    synchronized (this.singletonObjects) {
        Object singletonObject = this.singletonObjects.get(beanName);
        if (singletonObject == null) {
            if (this.singletonsCurrentlyInDestruction) {
                throw new BeanCreationNotAllowedException(beanName,
                        "Singleton bean creation not allowed while singletons of this factory
are in destruction " +
                        "(Do not request a bean from a BeanFactory in a destroy method
implementation!)");
            }
            if (logger.isDebugEnabled()) {
                logger.debug("Creating shared instance of singleton bean '" + beanName + "'");
            }
            beforeSingletonCreation(beanName);
            boolean newSingleton = false;
            boolean recordSuppressedExceptions = (this.suppressedExceptions == null);
            if (recordSuppressedExceptions) {
```

```
                this.suppressedExceptions = new LinkedHashSet<Exception>();
            }
            try {
                //这方法就是对上面第4步:bject beanInstance = doCreateBean(beanName, mbdToUse,
args);创建Bean的整体概述
                singletonObject = singletonFactory.getObject();
                newSingleton = true;
            }
            catch (IllegalStateException ex) {
                // Has the singleton object implicitly appeared in the meantime ->
                // if yes, proceed with it since the exception indicates that state.
                singletonObject = this.singletonObjects.get(beanName);
                if (singletonObject == null) {
                    throw ex;
                }
            }
            catch (BeanCreationException ex) {
                if (recordSuppressedExceptions) {
                    for (Exception suppressedException : this.suppressedExceptions) {
                        ex.addRelatedCause(suppressedException);
                    }
                }
                throw ex;
            }
            finally {
                if (recordSuppressedExceptions) {
                    this.suppressedExceptions = null;
                }
                afterSingletonCreation(beanName);
            }
            if (newSingleton) {
                //创建完成后 将创建的Bean添加到缓存中singletonObjects
                addSingleton(beanName, singletonObject);
            }
        }
        return (singletonObject != NULL_OBJECT ? singletonObject : null);
    }
}
```